Algorithm: Lab6 (By Sujiv Shrestha ID:610145)
Problem 1.

1. Show that any comparison-based algorithm to sort 4 elements requires at least 5 comparisons in the worst case.

Solution: For the comparison based algorithm to sort n elements we have seen that total comparisons required is O(logn!). So, for 4 elements in worst case,

$$\begin{aligned} \text{number of comparisons} &\geq \lceil \log 4! \rceil \\ &\geq \lceil \log 24 \rceil \\ &\geq \lceil 4.58 \rceil \\ &\geq 5 \end{aligned}$$

Problem2

2. Devise an algorithm that arranges the elements of a length-n integer array according to the following scheme:
position 0: the smallest integer
position 1: the largest integer
position 2: the second smallest integer
position 3: the second largest integer etc.
For example, this algorithm would arrange the input array {1, 2, 17, -4, -6, 8} as follows: {-6, 17, -4, 8, 1, 2}. (Notice that –6 is the smallest, 17 the largest, -4 second smallest, 8 second largest, etc.) Then answer the following:

Solution:

```
Algorithm alterSort(S)
    Input sequence S with n integers
    Output sequence Ans of n integers in alternate ascending and descending pattern
    dir←1
    for i←0 to n-1 do
        dir←0-dir
        tmp←0
        for j←0 to n-1-i do
            if(S[tmp]*dir<S[j]*dir) then
                tmp=j
        Ans[i]←S[tmp]
        S[tmp]=S[n-1-i]
    return Ans
```

Running Time , $T(n) = O(n^2)$

Java implementation of the Algorithm

```
public static int[] alterSort(int[] a) {
    int dir = 1;
    int[] ans = new int[a.length];
    for(int i=0;i<a.length;i++) {
        dir=0-dir;
```

```
                int tmp = 0;
                for(int j=0;j<a.length-i;j++) {
                        if(a[tmp]*dir<a[j]*dir) {
                                tmp=j;
                        }
                }
                ans[i]=a[tmp];
                a[tmp]=a[a.length-1-i];
        }
        return ans;
}
```

Note: *Other way of doing it is simply sort the sequence using quick sort or other faster method and swap the alternate terms of the sequence involving n/2 times of swap.*

A. What is the asymptotic running time of your algorithm?
Answer: The asymptotic running time of my algorithm is $O(n^2)$.

B. Prove that it is *impossible* to obtain a comparison-based algorithm to sort an integer array, in the way described above, that performs better than $\Theta(n \log n)$.
Answer: The comparison based sorting can sort a sequence of integer no faster than $\Theta(n \log n)$ and the desired sorting pattern in the question above require extra logic implemented during or after sorting so it is proved that it is impossible to obtain a comparison-based algorithm to sort an integer array in the way described above that performs better than $\Theta(n \log n)$.

Problem3

3. Carry out the steps of RadixSort to sort the following {80, 27, 72, 1, 27, 8, 64, 34, 16} – Hint: use 9 for your radix.
Solution:

Radix size = 9 [$\sqrt{80} \cong 9$]

Remainder: {80, 27, 72, 1, 27, 8, 64, 34, 16} % 9

27								
72	64						16	8
27	1						34	80
0	1	2	3	4	5	6	7	8

Quotient: {27,72,27,1,64,34,16,80,8} ÷ 9

			34					
8			27					80
1	16		27				64	72
0	1	2	3	4	5	6	7	8

Final result = {1,8,16,27,27,34,64,72,80} [Radix-sort is also stable]

Problem4

4. Describe an O(n) algorithm that does the following: Given an input array of *n* integers lying in the range 0 .. 3n - 1, the algorithm outputs the first integer that occurs in the array only once. (You may assume that each input array contains at least one number that has no duplicates in the array.) Explain why your algorithm has an O(n) running time.

Example: If the input array is [1, 2, 4, 9, 3, 2, 1, 4, 5], then the return value is 9 since 9 is the first integer that occurs in the array only once.

Solution:

Algorithm findOnce(S)
- **Input** sequence S with n integers that can range between 0 to 3n-1
- **Output** first integer in S that occurs only once in that array
- cnt←new array of size(3n-1)
- ans←new integer
- **for** i←0 to n-1 **do**
  - cnt[S[i]]← cnt[S[i]]+1
- **for** j←0 to n-1 **do**
  - **if**(cnt[S[j]]=1) **then**
    - ans←S[i]
    - **break**
- **return** ans

Running time, T(n) = O(n)

Java implementation of the algorithm

```
public static int findOnce(int[] s) {
	int[] cnt = new int[3*s.length-1];
	for(int i=0;i<s.length;i++) {
		cnt[s[i]]++;
	}
	for(int j=0;j<s.length;j++) {
		if(cnt[s[j]]==1)
			return s[j];
	}
	return 0;
}
```